ソフトウェア基礎科学：講義参考資料*

小林直樹

平成 23 年 1 月 21 日

0　数学の予備知識

ほとんどの人は大学の初等教育で習っているはずであるが，以下に本講義で頻繁に用いる数学の概念をまとめておく．

0.1　集合

（有限または無限個の）要素の集まりを集合と呼ぶ．

要素 $a_1, a_2, \ldots, a_n$ からなる集合を $\{a_1, a_2, \ldots, a_n\}$ と書く．$a_1, a_2, \ldots, a_n$ の並びは関係なく，例えば $\{a, b, c\}$ と $\{a, c, b\}$ は同じ集合である．また，要素が 0 個の集合を**空集合**とよび，$\emptyset$ または $\{\,\}$ で表す．

Example 0.1:

- $\{1, 2, 3\}$: 1, 2, 3 の 3 つからなう要素の集合
- $\{\emptyset, \{1\}\}$: 2 つの集合 $\emptyset$ と $\{1\}$ を要素とする集合

上の 2 番目のように，集合自身も別の集合の要素になることがある．（実際，集合論では，そのようにして自然数を含むすべての要素を構成する）

a が集合 X の要素であることを $a \in X$ と記す．逆に a が集合 X の要素でないときには $a \notin X$ と記す．例えば，$1 \in \{1, 2, 3\}$ だが $4 \notin \{1, 2, 3\}$.

集合 X の要素がすべて集合 Y の要素であるとき，X は Y の**部分集合**(subset) であるといい，$X \subseteq Y$ または $Y \supseteq X$ と書く．

0.1.1　集合上の演算

集合 X または Y の要素のみからなる集合を X と Y の**和集合**とよび，$X \cup Y$ と書く．例えば，$\{1, 4\} \cup \{2, 5\} = \{1, 2, 4, 5\}$.

集合 X と Y の両方に含まれる要素のみからなる集合を X と Y の**共通集合**とよび，$X \cap Y$ と書く．例えば，$\{1, 2, 4\} \cap \{2, 5\} = \{2\}$.

集合 X の部分集合すべてを要素としてもつ集合を X の**巾集合**(powerset) と呼び，2^X と書く．例えば，

$$2^{\{0,1,2\}} = \{\emptyset, \{0\}, \{1\}, \{2\}, \{0, 1\}, \{0, 2\}\{1, 2\}, \{0, 1, 2\}\}$$

*内容や記法は講義で扱うものとは必ずしも一致しないので注意.

上の定義から，$X \subseteq Y$ と $X \in 2^Y$ とは同値である．

集合 X の要素 x のうち，条件 $P(x)$ を満たすもののみからなる集合を $\{x \in X \mid P(x)\}$ と書く．例えば，

$$\{x \in \{0,1,2\} \mid x > 1\} = \{2\}$$

である．自然数の集合を **Nat** と書くことにすると，偶数のみからなる集合は，$\{x \in \mathbf{Nat} \mid \exists\ y \in \mathbf{Nat}.(2 \times y = x)\}$ と表すことができる．$\{x \in X \mid P(x)\}$ において，一番外側の $\exists$ は省略することが多い．したがって，$\{x \in \mathbf{Nat} \mid 2 \times y = x\}$ は $\{x \in \mathbf{Nat} \mid \exists\ y \in \mathbf{Nat}.(2 \times y = x)\}$ を意味し，$\{x \in \mathbf{Nat} \mid \forall\ y \in \mathbf{Nat}.(2 \times y = x)\}$ ではない．

集合 X の要素 x と集合 Y の要素 y の組 (x, y) からなる集合を X と Y の直積集合と呼び，$X \times Y$ で表す．例えば，$\{0,1\} \times \{2,3\} = \{(0,2),(0,3),(1,2),(1,3)\}$ である．同様に，$X_1, \ldots, X_n$ の要素 $x_1, \ldots, x_n$ の組 $(x_1, \ldots, x_n)$ からなる集合を $X_1 \times X_2 \times X_3 \times \cdots X_n$ と書く．[1]

0.2 関係

$X \times Y$ の部分集合を集合 X と Y の間の **2 項関係**と呼ぶ．$\mathcal{R}$ が集合 X と Y の間の **2 項関係**であるとき，$(x, y) \in \mathcal{R}$ が成り立つことを，しばしば $x\mathcal{R}y$ と記す．また，X と X の間の 2 項関係を X **上の 2 項関係**とも呼ぶ．

同様に，$X_1, \ldots, X_n$ の直積 $X_1 \times \cdots \times X_n$ の部分集合を $X_1, \ldots, X_n$ の間の n **項関係**と呼ぶ．

例えば，自然数の大小関係 $\leq$ は，

$$\begin{aligned} \leq = \{&(0,0),(0,1),(0,2),(0,3),\ldots,\\ &(1,1),(1,2),(1,3),\ldots,\\ &(2,2),(2,3),\ldots,\\ &\ldots\} \end{aligned}$$

という関係として定義される．

$\{(x, x) \mid x \in X\}$ を X 上の**恒等関係**と呼ぶ．

X 上の関係 $\mathcal{R}$ が以下の 3 つの条件をすべて満たすとき，$\mathcal{R}$ を X 上の**同値関係**と呼ぶ．

- **反射律**：任意の $x \in X$ について $(x, x) \in \mathcal{R}$.
- **対象律**：任意の $x, y \in X$ について $(x, y) \in \mathcal{R}$ ならば $(y, x) \in \mathcal{R}$.
- **推移律**：任意の $x, y, z \in X$ について $(x, y) \in \mathcal{R}$ かつ $(y, z) \in \mathcal{R}$ ならば $(x, z) \in \mathcal{R}$.

0.2.1 関係に関する演算

$\mathcal{R}_1 \subseteq X \times Y$, $\mathcal{R}_2 \subseteq Y \times Z$ であるとき，関係 $\mathcal{R}_1, \mathcal{R}_2$ の**合成** $\mathcal{R}_1 \circ \mathcal{R}_2$ を以下によって定義する．[2]

$$\{(x, z) \in X \times Z \mid \exists y.((x, y) \in \mathcal{R}_1 \wedge (y, z) \in \mathcal{R}_2)\}$$

例えば，$\mathcal{R}_1 = \{(1, a), (2, b)\}$，$\mathcal{R}_2 = \{(a, 3), (b, 2)\}$ であれば，$\mathcal{R}_1 \circ \mathcal{R}_2 = \{(1,3),(2,2)\}$ である．また，しばしば $\circ$ を省略して $\mathcal{R}_1\mathcal{R}_2$ と書く．

[1] 集合論では，組 (x, y) を $\{\{x\}, \{x, y\}\}$ で表す．

[2] $\mathcal{R}_2 \circ \mathcal{R}_1$ のように，$\mathcal{R}_1$ と $\mathcal{R}_2$ を逆に書く流儀もあるので注意．

$\mathcal{R}$ が X 上の 2 項関係であるとき，$\mathcal{R}^n$ を

$$\mathcal{R}^0 = \{(x, x) \mid x \in X\}$$
$$\mathcal{R}^{n+1} = \mathcal{R}^n\mathcal{R}$$

によって定義する. $\bigcup_{n\in\mathbf{Nat}} \mathcal{R}^n = \mathcal{R}^0 \cup \mathcal{R}^1 \cup \mathcal{R}^2 \cup \cdots$ を $\mathcal{R}$ の**反射推移的閉包**(reflexive transitive closure) と呼び，$\mathcal{R}^*$ と書く. 例えば，$\{(0,1),(1,2),(2,3)\}^* = \{(0,0),(0,1),(0,2),(0,3),(1,1),(1,2),(1,3),(2,2),(2,3),(3,3)\}$. $\mathcal{R}^*$ は，$\mathcal{R}$ を含み，反射律と推移律を満たす最小の関係に他ならない.

1 CCS

代表的な計算モデルとして，関数型プログラミング言語の基礎として用いられる λ 計算がある．しかし λ 計算は，複数の主体が互いに同期・通信を行いながら計算を進んでいくような並行プログラム・並行システムをモデル化するのには必ずしも適していない．並行プログラム・並行システムをモデル化するための計算モデルとして代表的なものに CCS, CSP, π 計算，Petri net などがある．ここでは Milner による CCS の枠組みをとりあげる.

CCS についてより詳しく知りたい人は

> Robin Milner, "*Communication and Concurrency*," Prentice Hall
> Robin Milner, "Communicating and mobile systems: the π-calculus," Cambridge University Press

等を参照。後者は 7 章までが CCS，8 章以降が π 計算について書かれている.

また、本資料後半で議論する様相 μ 計算およびモデル検査については

> Glynn Winskel, "The Formal Semantics of Programming Languages: An Introduction," Chapter 14, The MIT Press.

等を参照されたい．本資料の 1.5 節の記述はこの教科書を参考にしている.

1.1 CCS の基本 primitive

1.1.1 通信 primitive

CCS のもっとも基本的な primitive は他のプロセスとの送受信である。送受信は相手のプロセスを直接指定するかわりに通信路（以下チャネルと呼ぶ）を指定することによって行なう。送信プロセス・受信プロセスとも送信相手のプロセスがみつかるまで block する（より複雑な通信プロトコルはあとででてくる primitive との組合せによって実現できる)。

- $a.P\cdots$ チャネル a から受信をして P を実行する。
- $\overline{a}.P\cdots$ チャネル a に送信をして P を実行する。

Example 1.1: １００円玉しか受け付けない１５０円の切符の自動販売機 VM_1 は以下のように表すことができる。

$$VM_1 = coin_{100}.coin_{100}.\overline{change_{50}}.\overline{ticket}.VM_1$$

VM_1 は、100 円玉 2 枚を入れると 50 円玉のおつりと切符を出力するという動作を繰り返す。

1.1.2 choice

実際の世の中のシステムは、上の自動販売機の例のような一本道のやりとりではなく、さまざまな入出力の要求に対して対処できるようになっている。そこで、新たに + という operator を用いて複数の入出力から選択して実行するプロセスを表すことにする。例えば $a.P + \overline{b}.Q$ は、a からの入力あるいは b への出力が可能になるまで待ち、それにしたがって P または Q を実行するプロセスを表す。

これを用いて疑似的に通信時の値の受渡しを表すことができる。例えば a というチャネルを介して自然数 n を送信するプロセスは $\overline{a_n}.P$ によって、逆に受信するプロセスは $a_0.Q_0 + a_1.Q_1 + a_2.Q_2 + \cdots$ のように表すことができる。

Example 1.2: 途中で切符の購入を cancel できる自動販売機 VM_2 は以下のように表すことができる。

$$\begin{aligned} VM_2 = coin_{100}.(\quad & coin_{100}.(commit.\overline{change_{50}}.\overline{ticket}.VM_2 + cancel.\overline{change_{200}}.VM_2) \\ & + cancel.\overline{change_{100}}.VM_2) \end{aligned}$$

1.1.3 parallel composition

$P_1 \,|\, P_2$ によって「P_1 と P_2 を並行に実行する」ことを表す。

Example 1.3: 駅員がいて上の自動販売機 VM_2 を用いて切符を売っている状況を表してみよう。

$$\begin{aligned} VM_3 &= VM_2 \,|\, Man \\ Man &= \overline{mayI}.(\quad want_{ticket}.receive_{200}.\overline{coin_{100}}.\overline{coin_{100}}.\overline{commit}.change_{50}.ticket.\overline{give_{50\&ticket}}.\overline{thanks}.Man \\ &\qquad + question.\overline{answer}.Man) \end{aligned}$$

1.1.4 Restriction

上の駅員と自動販売機の組合せの例では、客が駅員の仲介をへずに勝手に自動販売機で切符を購入することが可能である。これを防ぐためには restriction という操作によって特定の channel が外に見えないようにすればよい。例えば **new** $a(a.P \,|\, \overline{a}.Q)$ とすると a というチャネルは外にはみえず、必ず $a.P$ と $\overline{a}.Q$ との間で通信がおこることを保証できる。

Example 1.4: 顧客から自動販売機が直接操作できないようにするには、

$$VM_4 = \textbf{new}\ coin_{100}, ticket, change_{50}, cancel, commit(VM_3)$$

とすればよい。

Exercise 1.5: VM_4 に対応する顧客の動作をプロセス式として表現せよ。

Example 1.6 [buffered communication]: primitive を用いた送信プロセス $\overline{a}.P$ は、a への送信が完了するまで P を実行できない。しかし、通信路自身をプロセスとして表現することによってさまざまな通信プロトコルを実現できる。例えば通信路として

$$Buffer = a_{Out}.\overline{a_{In}}.Buffer$$

を用いれば、出力プロセス $\overline{a_{Out}}.P$ は入力プロセス $a_{In}.Q$ を待たずに送信をすることができる。

Exercise 1.7: 上のプロセス $Buffer$ では、高だか一つしか先に送信を行えない。$Buffer$ を組み合わせて高だか n 回先に送信を行えるような n-place buffer を実現せよ。

1.2 Syntax of process expressions

上で導入したプロセス式の syntax をまとめると以下のようになる。

$$P ::= \mathbf{0} \mid a.P \mid \overline{a}.P \mid \tau.P \mid P_1 + P_2 \mid (P_1 \mid P_2) \mid \mathbf{new}\ aP \mid A[a_1, \ldots, a_n]$$

プロセス式全体の集合を $\mathcal{P}$ と書くことにする。$\mathbf{0}$ は何もしないプロセス、$\tau.P$ は内部遷移をしたのち P を実行するプロセス、A は $A[a_1, \ldots, a_n] = P$ の形のプロセス定義式によって定義される定数であり、$A[a_1, \ldots, a_n] = P$ のもとでは $A[b_1, \ldots, b_n]$ は、$[b_1/a_1, \ldots, b_n/a_n]P$ を表わす。以下では $A[\,]$ を単に A とかく。$\mathbf{new}\ a_1 \cdots \mathbf{new}\ a_n P$ をしばしば $\mathbf{new}\ a_1, \ldots, a_n P$ と略記する.

Example 1.8: τ 遷移を用いることによって unreliable な network を次のようなプロセス式として表現できる。

$$U = a_{Out}.(\overline{a_{In}}.U + \tau.U)$$

Exercise 1.9: unreliable な network で用いられるプロトコルの一つである alternating bit protocol をプロセス式として表現せよ。

Example 1.10: 1bit の共有メモリは、次のようなプロセスとして実現できる。

$$Bit_0 = write_0.Bit_0 + write_1.Bit_1 + \overline{read_0}.Bit_0$$
$$Bit_1 = write_0.Bit_0 + write_1.Bit_1 + \overline{read_1}.Bit_1$$

Exercise 1.11: 共有メモリを介して 2 つのプロセス間の mutual exclusion を実現するアルゴリズムを CCS のプロセス式として表現せよ。

1.3 Operational semantics

操作的意味の与え方にはいくつかの方法があるが、ここでは labelled transition semantics と呼ばれる意味論を用いる。

「P は α という action をして Q に遷移する」を表わす 3 項関係 $P \xrightarrow{\alpha} Q$ を以下のルールによって inductive に定義する。ただし action α とは内部遷移 τ あるいは入力 a, 出力 $\overline{a}$ のいずれかである。$\overline{\alpha}$ を $\alpha = a$ のとき $\overline{\alpha} = \overline{a}$, $\alpha = \overline{a}$ のとき $\overline{\alpha} = a$ によって定義する。

$$a.P \xrightarrow{a} P \qquad \text{(CCS-In)}$$

$$\overline{a}.P \xrightarrow{\overline{a}} P \qquad \text{(CCS-Out)}$$

$$\tau.P \xrightarrow{\tau} P \qquad \text{(CCS-}\tau\text{)}$$

$$\frac{P \xrightarrow{\alpha} Q}{P + R \xrightarrow{\alpha} Q} \qquad \text{(CCS-SumL)}$$

$$\frac{P \xrightarrow{\alpha} Q}{R + P \xrightarrow{\alpha} Q} \qquad \text{(CCS-SumR)}$$

$$\frac{P \xrightarrow{\alpha} Q}{P \mid R \xrightarrow{\alpha} Q \mid R} \qquad \text{(CCS-ParL)}$$

$$\frac{P \xrightarrow{\alpha} Q}{R \,|\, P \xrightarrow{\alpha} R \,|\, Q} \qquad \text{(CCS-ParR)}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'} \qquad \text{(CCS-Com)}$$

$$\frac{P \xrightarrow{\alpha} Q \qquad \alpha \notin \{a_1, \ldots, a_n, \overline{a_1}, \ldots, \overline{a_n}\}}{\textbf{new } a_1, \ldots, a_n P \xrightarrow{\alpha} \textbf{new } a_1, \ldots, a_n Q} \qquad \text{(CCS-Restr)}$$

$$\frac{[b_1/a_1, \ldots, b_n/a_n]P \xrightarrow{\alpha} Q \qquad A[a_1, \ldots, a_n] = P}{A[b_1, \ldots, b_n] \xrightarrow{\alpha} Q} \qquad \text{(CCS-Def)}$$

Exercise 1.12: VM_1, VM_2 の遷移を上の関係によって表わせ。

Exercise 1.13: 任意の P_1, P_2, P_3, Q, α について $(P_1 + P_2) + P_3 \xrightarrow{\alpha} Q \Leftrightarrow P_1 + (P_2 + P_3) \xrightarrow{\alpha} Q$ が成り立つことを示せ。

Exercise 1.14: 上のルールにおいて (CCS-SumL), (CCS-SumR) の代わりに次の 2 つのルールを採用した場合、何か違いが生じるか？

$$P + Q \xrightarrow{\tau} P \qquad \text{(CCS-SumL')}$$

$$P + Q \xrightarrow{\tau} Q \qquad \text{(CCS-SumR')}$$

Example 1.15 [alternating bit protocol]: Alternating bit protocol は、unreliable な network を介して安全にデータを送るための protocol である。データを送信するたびに交互に 0 か 1 の acknowledgement を待ち、ack がこなければ再びデータを送信しなおす。sender, receiver 側のプロセスは次のように記述できる。

$$Accept_b = \Sigma_i accept_i.Send_{i,b}$$
$$Send_{i,b} = \overline{send_{i,b}}.Sending_{i,b}$$
$$Sending_{i,b} = (\tau.Send_{i,b} + ack_b.Accept_{\hat{b}} + ack_{\hat{b}}.Sending_{i,b})$$

$$Receive_b = \Sigma_i(receive_{i,b}.\overline{deliver_i}.Reply_b) + \Sigma_i(receive_{i,\hat{b}}.Reply_{\hat{b}}) + \tau.Reply_{\hat{b}}$$
$$Reply_b = \overline{reply_b}.Receive_{\hat{b}}$$

ただし b は 0 または 1 であり、$\hat{b}$ は $1 - b$ を表す。$\Sigma_i P_i$ は、$P_1 + \cdots + P_n$ の略である。

Exercise 1.16: 上で示した alternating bit protocol を実現するプロセスについて、$send, receive$ をつなぐ unreliable な送信用ネットワーク、$reply, ack$ をつなぐ unreliable な返答用ネットワークをプロセスとして記述せよ。それらのプロセスを $SendNet, ReplyNet$ としたとき、

$$\textbf{new } send_{i,b}, ack_b, receive_{i,b}, reply_b (i, b \in \{0, 1\})(Accept_0 \,|\, SendNet \,|\, ReplyNet \,|\, Receive_0)$$

はどのようなプロセスとしてみえるか述べよ。

Exercise 1.17: Example 1.15 では、受信側は、実際に ($deliver$ をとおして）client に値を受け渡すまで acknowledgment を送信しないため、送信側が無駄なメッセージを何度もおくりなおす可能性がある。client が受信可能になる前に acknowledgment を送信するように receiver 側のプロセスを変更せよ。変更後の client からみた全体のふるまいは、もとのそれと同じか？

1.4 Process Equivalence

Exercise 1.16 で構成した alternating bit protocol が正しく動作することをいうためには、プロセスが全体としてただの buffer を表すプロセス

$$Buffer = \Sigma_i(accept_i.\overline{deliver_i}.Buffer)$$

であるかのように振舞うことを示せばいいと考えられる。

また、Exercise 1.7 では、容量 1 の buffer を n 個つなぐことで容量 n の buffer が構成できることをみたが、これをきちんと確かめるためには、容量 n の buffer とは何かをプロセス式として定義し、容量 1 の buffer を n 個つなげたものが実際にそれと同じ振舞いをすることを示す必要があるだろう。

ではいったいどういう時に「プロセス P がプロセス Q と同じ振舞いをする」といえるのだろうか？これをきちんと定義しないかぎり、上の alternating bit protocol や容量 n の buffer について「なんとなく正しそうだ」以上のことは言えない。

1.4.1 trace equivalence

プロセスの等価性の議論は当然、操作的意味を与える $P \xrightarrow{\alpha} Q$ という関係に基いて行なう必要がある。$P \xrightarrow{\alpha} Q$ は、「P は α という action をして Q に遷移する」ことを表しているので、オートマトンの遷移に似ている。そこで、オートマトンの等価性にならって、可能な action 列の集合

$$Tr(P) = \{\alpha_1 \cdots \alpha_n \mid \exists Q_1.\cdots\exists Q_n.(P \xrightarrow{\alpha_1} Q_1 \xrightarrow{\alpha_2} Q_2 \cdots \xrightarrow{\alpha_n} Q_n)\}$$

を考え、$Tr(P) = Tr(Q)$ のとき、プロセス P とプロセス Q は等価であると考えたらどうであろうか？すると、例えば

- $P = \overline{a}.b.\mathbf{0}$ と $Q = \overline{a}.b.\mathbf{0} + \overline{c}.\mathbf{0}$ は $\overline{c} \in Tr(Q)$ であるが $\overline{c} \notin Tr(P)$ なので P と Q は異なる。
- $P = \overline{a}.\mathbf{0}|\overline{b}.\mathbf{0}$ と $Q = \overline{a}.\overline{b}.\mathbf{0} + \overline{b}.\overline{a}.\mathbf{0}$ は、

  $$Tr(P) = Tr(Q) = \{\epsilon, \overline{a}, \overline{b}, \overline{a}\overline{b}, \overline{b}\overline{a}\}$$

  となり、P と Q は等しい。

と判定できる。一般にこのような等価性を trace equivalence と呼ぶ。これは「同じ言語を受理する」というオートマトンの等価性の判定条件に相当する。

ところが、このような基準は並行プロセスの等価性の基準としては必ずしも十分でないことがわかる。例えば

$$P = a.(\overline{b}.\mathbf{0} + \overline{c}.\mathbf{0})$$
$$Q = a.\overline{b}.\mathbf{0} + a.\overline{c}.\mathbf{0}$$

を考えよう。すると、

$$Tr(P) = Tr(Q) = \{\epsilon, a, a\overline{b}, a\overline{c}\}$$

であるので、trace equivalence の基準では P と Q は等しい。ところが、P, Q をプロセス $R = \overline{a}.b.\mathbf{0}$ と並行に走らせると、$Q \mid R$ は、a に関して通信した際

$$Q \mid R \xrightarrow{\tau} \overline{c}.\mathbf{0} \mid b.\mathbf{0}$$

となってそれ以上単独では通信ができなくなる可能性があるのに対し、$P \mid R$ は

$$P \mid R \xrightarrow{\tau} \overline{b}.\mathbf{0} + \overline{c}.\mathbf{0} \mid b.\mathbf{0}$$

となり、a の通信後つねに b に関する通信が可能である。したがって、P と Q は同一視することができない。

1.4.2 Bisimulation

上でででてきたプロセス

$$P = a.(\overline{b}.\mathbf{0} + \overline{c}.\mathbf{0})$$
$$Q = a.\overline{b}.\mathbf{0} + a.\overline{c}.\mathbf{0}$$

をきちんと区別するためにはどのように等価性を定義すればよいだろうか？P と Q が異なるのは、a という同じ action をしたあとで前者は常に $\overline{b}$ という action が可能であるにかかわらず、後者はそのような action ができない可能性があるからである。したがって、P と Q が等しいとは

- P がある action ができるならば Q もその action ができ、かつ遷移後のプロセス同士について同じことがいえる。
- Q がある action ができるならば P もその action ができ、かつ遷移後のプロセス同士について同じことがいえる。

とすればよさそうである。
これを formal に定義すると以下のようになる。

Definition 1.18 [bisimulation]: プロセス式の 2 項関係 $\mathcal{R}$ が *(strong) bisimulation* であるとは、任意の $(P, Q) \in \mathcal{R}$ および action α に対し、

- $P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.(Q \xrightarrow{\alpha} Q' \wedge (P', Q') \in \mathcal{R})$
- $Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.(P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R})$

が成り立つことをいう。最大の bisimulation を $\sim$ と書き、$(P, Q) \in \sim$ のとき $P \sim Q$ とかく。

Exercise 1.19: $\sim$ が同値関係であること、すなわち

- reflexivity: $\forall P.(P \sim P)$
- symmetricity: $\forall P, Q.(P \sim Q \Rightarrow Q \sim P)$
- transitivity: $\forall P, Q, R.(P \sim Q \wedge Q \sim R \Rightarrow P \sim R)$

を満たすことを示せ。

Exercise 1.20: 2 つのプロセス

$$P = a.(\overline{b}.\mathbf{0} + \overline{c}.\mathbf{0})$$
$$Q = a.\overline{b}.\mathbf{0} + a.\overline{c}.\mathbf{0}$$

について、$P \not\sim Q$ であることを示せ。

Exercise 1.21: 2 つのプロセス

$P = a.\mathbf{0} \,|\, b.\mathbf{0}$
$Q = a.b.\mathbf{0} + b.a.\mathbf{0}$

について、$P \sim Q$ であることを示せ。

Exercise 1.22: 任意の P, Q, R について

- $P \,|\, \mathbf{0} \sim P$
- $P \,|\, Q \sim Q \,|\, P$
- $(P \,|\, Q) \,|\, R \sim P \,|\, (Q \,|\, R)$

が成り立つことを示せ。

Exercise 1.23: 1 または 2 の値を送信する buffer process:

$$\begin{array}{lcl} Buffer[in_1, in_2, out_1, out_2] & = & in_1.\overline{out_1}.Buffer[in_1, in_2, out_1, out_2] + in_2.\overline{out_2}.Buffer[in_1, in_2, out_1, out_2] \\ TwoBuffer & = & in_1.TwoBuffer_1 + in_2.TwoBuffer_2 \\ TwoBuffer_1 & = & in_1.\overline{out_1}.TwoBuffer_1 + in_2.\overline{out_1}.TwoBuffer_2 + \overline{out_1}.TwoBuffer \\ TwoBuffer_2 & = & in_1.\overline{out_2}.TwoBuffer_1 + in_2.\overline{out_2}.TwoBuffer_2 + \overline{out_2}.TwoBuffer \end{array}$$

について、

$$TwoBuffer \sim \mathbf{new}\ c_1, c_2(Buffer[in_1, in_2, c_1, c_2] \,|\, Buffer[c_1, c_2, out_1, out_2])$$

は成り立つか？

alternative formulation of bisimulation strong bisimulation $\sim$ は、以下のように定義される関数

$$\mathcal{F} \in 2^{\mathcal{P}\times\mathcal{P}} \to 2^{\mathcal{P}\times\mathcal{P}}$$

の greatest fixpoint として定義することもできる。すなわち

$$\begin{array}{rl} \mathcal{F}(\mathcal{R}) = \{(P,Q) \mid & \forall\alpha, P'.(P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.(Q \xrightarrow{\alpha} Q' \wedge (P', Q') \in \mathcal{R})) \\ & \wedge \forall\alpha, Q'.(Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.(P \xrightarrow{\alpha} P' \wedge (P', Q') \in \mathcal{R}))\} \end{array}$$

によって $\mathcal{F}$ を定義すると $\mathcal{F}$ は明らかに単調関数であり、

$$\sim = \bigcup\{S \subseteq \mathcal{F}(S)\} \subseteq \bigcap_{n\in\omega} \mathcal{F}^n(\mathcal{P}\times\mathcal{P})$$

が成り立つ。したがって、

- $P \sim Q$ を示すにはある集合 $S \ni (P,Q)$ が存在して $S \subseteq \mathcal{F}(S)$ が成り立つことを、
- $P \not\sim Q$ を示すにはある自然数 n について $(P,Q) \notin \mathcal{F}^n(\mathcal{P}\times\mathcal{P})$ が成り立つことを

示せばよいことがわかる。

Exercise 1.24: $\bigcup\{S \subseteq \mathcal{F}(S) \mid S \subseteq \mathcal{P}\times\mathcal{P}\} \subseteq \bigcap_{n\in\omega} \mathcal{F}^n(\mathcal{P}\times\mathcal{P})$ を示せ。（$\supseteq$ は一般には成り立たない。）

1.4.3 Weak Bisimulation (Observational Equivalence)

Exercise 1.23 からもわかるとおり、strong bisimulation は、プロセスの等価性の基準としてはきつすぎる。例えば $P = \tau.\mathbf{0}$ を考えると、内部遷移は本来外からは観測できないものなので、P は $\mathbf{0}$ と区別できないはずである。ところが strong bisimulation の基準では $P \not\sim \mathbf{0}$ である。そこで、bisimulation の定義を変更して、内部遷移 $\xrightarrow{\tau}$ を観測できないようにしよう。

まず 2 項関係 $\stackrel{\alpha}{\Rightarrow} \subseteq \mathcal{P} \times \mathcal{P}$ を

- $\alpha \neq \tau$ のとき、$\xrightarrow{\tau}{}^{*} \xrightarrow{\alpha} \xrightarrow{\tau}{}^{*}$
- $\stackrel{\tau}{\Rightarrow} = \xrightarrow{\tau}{}^{*}$

によって定義する。ただし $\xrightarrow{\tau}{}^{*}$ は $\xrightarrow{\tau}$ の reflexive transitive closure である。

Definition 1.25 [weak bisimulation]: プロセス式の 2 項関係 $\mathcal{R}$ が *weak bisimulation* であるとは、任意の $(P, Q) \in \mathcal{R}$ および action α に対し、

- $P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.(Q \stackrel{\alpha}{\Rightarrow} Q' \wedge (P', Q') \in \mathcal{R})$
- $Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.(P \stackrel{\alpha}{\Rightarrow} P' \wedge (P', Q') \in \mathcal{R})$

が成り立つことをいう。最大の weak bisimulation を $\approx$ と書き、$(P, Q) \in \approx$ のとき $P \approx Q$ とかく。

Exercise 1.26: $\approx$ が同値関係であることを示せ。

Exercise 1.27: 任意のプロセス P について $\tau.P \approx P$ が成り立つことを示せ。

あるプロセス P, Q について $P \approx Q$ あるいは $P \not\approx Q$ であることを示すには次の定理が便利である。

Theorem 1.28:

$$
\begin{aligned}
&P \approx Q \Leftrightarrow \\
&\quad \forall \alpha, P'.(P \xrightarrow{\alpha} P' \Rightarrow \exists Q'.(Q \stackrel{\alpha}{\Rightarrow} Q' \wedge P' \approx Q')) \\
&\quad \wedge \forall \alpha, Q'.(Q \xrightarrow{\alpha} Q' \Rightarrow \exists P'.(P \stackrel{\alpha}{\Rightarrow} P' \wedge P' \approx Q'))
\end{aligned}
$$

Exercise 1.29: Theorem 1.28 を証明せよ。

Exercise 1.30: $\tau.a.\mathbf{0} + \overline{b}.\mathbf{0} \not\approx a.\mathbf{0} + \overline{b}.\mathbf{0}$ を示せ。

Exercise 1.31: Exercise 1.23 のプロセスについて

$$TwoBuffer \approx \mathbf{new}\ c_1, c_2(Buffer[in_1, in_2, c_1, c_2] \mid Buffer[c_1, c_2, out_1, out_2])$$

を示せ。

Exercise 1.32: Exercise 1.16 のプロセスについて

$$
\begin{aligned}
&\mathbf{new}\ send_{i,b}, ack_b, receive_{i,b}, reply_b(Accept_0 \mid SendNet \mid ReplyNet \mid Receive_0) \\
&\approx Buffer
\end{aligned}
$$

が成り立つかどうか判定せよ。ただし、i のとりうる値は 1 または 2 とし、$Buffer$ は以下のように定義する。

$$Buffer = accept_1.\overline{deliver_1}.Buffer + accept_2.\overline{deliver_2}.Buffer$$

Exercise 1.33: $P \approx Q \Rightarrow P \mid R \approx Q \mid R$ を示せ。

1.4.4 Observational Congruence

Exercise 1.30 からわかるように、$\approx$ は congruence relation ではない。すなわち $P \approx Q$ であってもあるプロセス中の任意の P の出現を Q で置き換えてよいわけではない。congruence relation を得るには、weak bisimulation の最初の遷移の条件だけを若干強めればよい。$\stackrel{\alpha}{\Rightarrow}_c = \stackrel{\tau}{\rightarrow}^* \stackrel{\alpha}{\rightarrow} \stackrel{\tau}{\rightarrow}^*$ とする。

Definition 1.34 [observational congruence]: P と Q が observationally congruence であるとは、任意の $(P, Q) \in \mathcal{R}$ および action α に対し、

- $P \stackrel{\alpha}{\rightarrow} P' \Rightarrow \exists Q'.(Q \stackrel{\alpha}{\Rightarrow}_c Q' \wedge P' \approx Q')$
- $Q \stackrel{\alpha}{\rightarrow} Q' \Rightarrow \exists P'.(P \stackrel{\alpha}{\Rightarrow}_c P' \wedge P' \approx Q')$

が成り立つことをいう。このとき、$P \approx_c Q$ とかく。

Exercise 1.35: $P \approx_c Q \Rightarrow P + R \approx_c Q + R$ を示せ。

1.5 Process Specification Language

1.5.1 Introduction

前節ではプロセスの等価性の判定によって、プロセスが期待どおりのふるまいをするかどうかのチェックができることをみた。しかし、プロセスが期待されるふるまいをするかどうかを論じるには、等価性の判定の基準は厳しすぎることが多い。例えば、

> 「はじめに a というチャネルから受信をしたら次のステップで b というチャネルに送信できなければならない。それ以外はどんなふるまいをしてもよい」

という性質は、あるプロセスとの等価性という基準からは直接記述することができない（実際、互いに等価ではない無限の数のプロセスが上の性質を満たす）。そこで、プロセスの仕様を記述するために、次のような式を導入しよう。

- $\langle\alpha\rangle A \cdots$「$\alpha$ という action を行い、その後 A という条件を満たすことが可能である」
- $[\alpha]A \cdots$「α という action を行ったあとは常に A という条件を満たす」。

すると上のプロセスの条件は、$[a]\langle b\rangle\mathbf{true}$ と書くことができる。

Exercise 1.36: 次の式はどのような条件を表わすか推測せよ。

1. $[a]\mathbf{false}$
2. $\langle a\rangle\langle b\rangle([c]\mathbf{false} \wedge [d]\mathbf{false})$

Negation $\neg A$ を導入すると $[\alpha]A$ は $\neg\langle\alpha\rangle\neg A$ と表わすことができる。

上で導入したプロセスの仕様を表わす式の syntax は以下のようにまとめられる。

$$A ::= \mathbf{true} \mid A_1 \wedge A_2 \mid \neg A \mid \langle\alpha\rangle A$$

プロセス P が仕様 A を満たすという関係 $P \models A$ は以下の条件を満たす最小の関係として定義できる。

$$
\begin{array}{lll}
P \models \mathbf{true} & & \text{for any } P \\
P \models A_1 \wedge A_2 & \text{if} & P \models A_1 \text{ and } P \models A_2 \\
P \models \neg A & \text{if} & P \models A \text{ does not hold} \\
P \models \langle\alpha\rangle A & \text{if} & \exists Q.(P \xrightarrow{\alpha} Q)
\end{array}
$$

Exercise 1.37: $[\alpha]A = \neg\langle\alpha\rangle\neg A$, $\mathbf{false} = \neg\mathbf{true}$ とするとき、次の条件が成り立つことを示せ。

1. $P \models [\alpha]A \Leftrightarrow \forall Q.(P \xrightarrow{\alpha} Q \Rightarrow Q \models A)$
2. $P \models [\alpha]\mathbf{false} \Leftrightarrow P \xrightarrow{\alpha} Q$ を満たす Q は存在しない

Exercise 1.38: 仕様を表わす式を次のように拡張する。

$$A ::= \bigwedge_{i\in I} A_i \mid \neg A \mid \langle\alpha\rangle A$$

$P \models \bigwedge_{i\in I} A_i$ を

$$P \models \bigwedge_{i\in I} A_i \Leftrightarrow P \models A_i \text{ for any } i \in I$$

によって定義する。このとき、

$$P \sim Q \Leftrightarrow \forall A.(P \models A \Leftrightarrow Q \models A)$$

が成り立つことを示せ。

1.5.2 不動点演算子

上で導入した式では、プロセスの有限ステップの遷移に関する仕様しか記述することができない。例えば「何回か内部遷移をしたあとで a に送信を行える状態になる可能性がある」という性質を有限の $\wedge, \neg, \langle\alpha\rangle, [\alpha]$ の組み合わせで記述するのは不可能である。

そこで新しい論理演算子を導入することにしよう。上の仕様は、直観的には

$$\langle a\rangle\mathbf{true} \vee \langle\tau\rangle\langle a\rangle\mathbf{true} \vee \langle\tau\rangle\langle\tau\rangle\langle a\rangle\mathbf{true} \vee \cdots$$

という無限の式で表わすことができる。この式は、$\langle a\rangle\mathbf{true} \vee \langle\tau\rangle X \Leftrightarrow X$ を満たす（X をそれを満たすプロセスの集合としてとらえたときに）最小の X と考えることができる。そこで、これを新しい operator μ を導入して

$$\mu X.(\langle a\rangle\mathbf{true} \vee \langle\tau\rangle X)$$

と書くことにする。

さらに、「ある action をして A を満たす状態になることができる」ことを表わすために、$\langle . \rangle A$ という式を導入する。すると、「A を満たす状態に遷移するかもしれない」ということを表わす式は

$$possibly(A) = \mu X.(A \vee \langle . \rangle X)$$

と表わすことができる。

Exercise 1.39: $\neg\langle . \rangle\neg A$ がどのような条件を表すかを述べよ。

以下では $\neg\langle . \rangle\neg A$ を $[.]A$ と書くことにする。

Example 1.40: 「遷移の過程で常に B という条件が満たされる」という条件は

$$\neg\mu X.(\neg B \vee \langle . \rangle X)$$

と表すことができる。

以下 $\neg\mu X.\neg[\neg X/X]A$ を $\nu X.A$ と書くことにする。上の $\neg\mu X.(\neg B \vee \langle . \rangle X)$ は $\nu X.(B \wedge [.]X)$ と書くことができる。

Exercise 1.41: 次の式がどのような条件を表わすか推測せよ。

- $\mu X.(\langle a \rangle \mathbf{true} \vee [.]X)$
- $\mu X.(A \vee (\langle . \rangle \mathbf{true} \wedge [.]X)$

Exercise 1.42: 正常に終了した状態を表わす式を $Terminal$ と表わすことにする。「正常終了でないのに何も action ができなくなる状態（deadlock) に陥る可能性がある」ことを表わす式を書け。

1.5.3 Syntax and semantics of the modal μ-calculus

上で導入したような体系を modal μ-calculus（あるいは modal ν-calculus) と呼ぶ。syntax をまとめると以下のようになる。（あとの都合上、ここでは ν の方を primitive とする）

$$A ::= S \mid X \mid \mathbf{true} \mid A \wedge B \mid \neg A \mid \langle \alpha \rangle A \mid \langle . \rangle A \mid \nu X.A$$

S はプロセスの集合を表わし、X は ν によって bind される変数を表す。$\nu X.A$ において、X は A 中の positive な位置（偶数回の negation が適用されている位置）のみに出現するものとする。以下では closed な式、すなわち変数の free な出現を含まない式のみを考える。

「プロセス P が A を満たす」という関係 $P \models A$ を厳密に定義するため、以下で A の意味 A^* をプロセスの集合として与え、$P \models A \Leftrightarrow P \in A^*$ とする。A^* は以下のように定義できる（Pr はプロセス全体の集合)。

$$\begin{array}{lcl}
S^* & = & S \\
\mathbf{true}^* & = & \mathrm{Pr} \\
(A \wedge B)^* & = & A^* \cap B^* \\
(\neg A)^* & = & \mathrm{Pr} \setminus A^* \\
(\langle \alpha \rangle A)^* & = & \{P \mid \exists Q \in A^*.(P \xrightarrow{\alpha} Q)\} \\
(\langle . \rangle A)^* & = & \{P \mid \exists Q \in A^*.\exists \alpha.(P \xrightarrow{\alpha} Q)\} \\
(\nu X.A)^* & = & \bigcup\{S \subseteq \mathrm{Pr} \mid S \subseteq ([S/X]A)^*\}
\end{array}$$

Exercise 1.43: 上の定義が well defined であること、すなわち任意の A について $A^* = S$ を満たす $S \subseteq \mathrm{Pr}$ がただ一つ存在することを示せ。

Exercise 1.44: $(\nu X.A)^*$ が $S = ([S/X]A)^*$ を満たす最大の集合であることを示せ。

Exercise 1.45: $(\neg\nu X.\neg[\neg X/X]A)^* = \bigcap\{S \subseteq \mathrm{Pr} \mid ([S/X]A)^* \subseteq S\}$ を示せ。

Exercise 1.46: $\nu X.(B \vee (A \wedge [.]X))$ はどのような条件を表わすか述べよ。

1.5.4 Model checking

本節では、$P \models A$ が成り立つかどうかをチェックする方法を説明する。

$P \models A$ をチェックする場合に一番問題となるのは greatest fixpoint operator ν の扱いである。$\nu X.P$ という形の formula の存在のために A^* を直接計算することはできない。しかし、$P \models \nu X.A$ をチェックするためには、$(\nu X.A)^*$ 自身を計算する必要はなく、P から到達可能なプロセスの集合

$$reach(P) = \{Q \in \mathrm{Pr} \mid P \dot{\rightarrow}^* Q\}$$

($P \dot{\rightarrow} Q$ は $\exists\alpha.(P \xrightarrow{\alpha} Q)$ によって定義される 2 項関係) について $(\nu X.A)^* \cap reach(P)$ を計算すれば十分である。$reach(P)$ が有限集合の場合にはこれが実際に計算可能であることは以下のようにしてわかる。

$P \in \mathrm{Pr}$ にたいし、$A|_P$ を次のように定義する。

$$\begin{array}{lcl}
S|_P & = & S \cap reach(P) \\
\mathbf{true}|_P & = & reach(P) \\
(A_1 \wedge A_2)|_P & = & A_1|_P \wedge A_2|_P \\
(\neg A)|_P & = & reach(P) \backslash A|_P \\
(\langle\alpha\rangle A)|_P & = & \{Q \in reach(P) \mid \exists R \in reach(P).(Q \xrightarrow{\alpha} R \text{ and } R \in A|_P)\} \\
(\langle . \rangle A)|_P & = & \{Q \in reach(P) \mid \exists R \in reach(P).\exists\alpha.(Q \xrightarrow{\alpha} R \text{ and } R \in A|_P)\} \\
(\nu X.A)|_P & = & \bigcup\{S \subseteq reach(P) \mid S \subseteq ([S/X]A)|_P\}
\end{array}$$

Lemma 1.47: $A|_P = A^* \cap reach(P)$

Exercise 1.48: 上の補題を証明せよ。

上の補題により、$P \models A$ を判定するためには、P が $A|_P$ の要素であるか否かを判定すればよい. $reach(P)$ が有限集合の場合には定義に従って実際に $A|_P$ が計算可能なので (($(\nu X.A)|_P$ の定義おいて、$S \subseteq reach(P)$ を満たす S を列挙できることに注意)、$P \models A$ は決定可能であることがわかる.

ただし、実際に $(\nu X.A)|_P$ を定義に従って計算するのはコストがかかるので、以下のような不動点の性質を用いて計算を行う。

Theorem 1.49: A_n を $A_0 = \mathbf{true}, A_{i+1} = [A_i/X]A$ によって定義される式とする。$([\mathbf{true}/X]A_n)|_P$ は n に関して単調に減少し、集合 $reach(P)$ が有限であれば、ある n が存在して

$$(\nu X.A)|_P = ([\mathbf{true}/X]A_n)|_P$$

が成り立つ。

Exercise 1.50: 上の定理を証明せよ。

Exercise 1.51: P を次によって定義されるプロセスとする。

$$P = a.P + \tau.Q$$
$$Q = b.Q$$

また、A を $\nu X.(\langle b\rangle\mathbf{true} \vee (\langle a\rangle\mathbf{true} \wedge [.]X)$ とする。$A|_P$ を計算することにより、$P \models A$ が成り立つか否かを判定せよ。

Exercise 1.52: mutual exclusion を実現する (Peterson による) アルゴリズム:

```
repeat
    flag[i]:=true;
    turn := j;
    while (flag[j] and turn=j) do skip;
    ******************
    critical sections
    ******************
    flag[i]:=false;
until false;
```

は次のように記述できる。

$$
\begin{aligned}
&Bit_b[read_0, read_1, write_0, write_1] \\
&\quad = \overline{read_b}.Bit_b[read_0, read_1, write_0, write_1] \\
&\qquad + write_0.Bit_0[read_0, read_1, write_0, write_1] + write_1.Bit_1[read_0, read_1, write_0, write_1] \\
&P_b = \overline{setf_b}.\overline{setT_{\hat{b}}}.Wait_i \\
&Wait_b = readf_{\hat{b},1}.(readt_{\hat{b}}.Wait_b + readt_b.CR_b) + readf_{\hat{b},0}.CR_b \\
&CR_b = \overline{cr_b}.\overline{rsetf_b}.P_b \\
&System = \mathbf{new}\ C(Bit_0[readt_0, readt_1, setT_0, setT_1] \\
&\qquad\qquad | Bit_0[readf_{0,0}, readf_{0,1}, rsetf_0, setf_0] \\
&\qquad\qquad | Bit_1[readf_{1,0}, readf_{1,1}, rsetf_1, setf_1] \\
&\qquad\qquad | P_0 | P_1) \\
&C = readt_0, readt_1, readf_{0,0}, readf_{0,1}, readf_{1,0}, readf_{1,1}, setT_0, setT_1, setf_0, setf_1, rsetf_0, rsetf_1
\end{aligned}
$$

以下の問に答えよ。

- 「critical section が同時に実行されることはない」ことをいうためには System がどのような assertion A を満たすことを示せばよいか？System が実際にその式を満たしているかどうかチェックせよ。
- 「P_0, P_1 とも critical section を実行せずに永遠に wait することはない」ことをいうためには System の定義を若干変更する必要がある。System をどのように変更し、変更したプロセスがどのような assertion A を満たせばよいかを述べよ。